Integrating Marine Observatories into a System-of-Systems: Messaging in the US Ocean Observatories Initiative

Matthew Arrott[1], Alan D. Chave[2], Claudiu Farcas[1], Emilia Farcas[1], Jack E. Kleinert[3], Ingolf Krueger[1], Michael Meisinger[1], John A. Orcutt[4], Cheryl Peach[4], Oscar Schofield[5], Munindar P. Singh[6] and Frank L. Vernon[4]
[1] Calit2, University of California at San Diego, La Jolla, CA 92093-0436, USA
[2] Woods Hole Oceanographic Institution, Woods Hole, MA 02543, USA
[3] Raytheon Intelligence and Information Systems, Aurora, CO 80011, USA
[4] Scripps Institution of Oceanography, La Jolla, CA 92093, USA
[5] COOL, Rutgers University, New Brunswick, NJ 08901, USA
[6] Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA

Abstract- **The Ocean Observatories Initiative (OOI) will implement ocean sensor networks covering a diversity of oceanic environments, ranging from the coastal to the deep ocean. Construction will begin in Fall 2009, with deployment phased over five years. The integrating feature of the OOI is a comprehensive Cyberinfrastructure (CI), whose design is based on loosely-coupled distributed services, and whose elements are expected to reside throughout the physical components; from seafloor instruments to autonomous vehicles to deep sea moorings to shore facilities to computing and storage infrastructure. The OOI-CI provides novel capabilities for data acquisition, distribution, modeling, planning and interactive control of oceanographic experiments. The architecture comprises six subsystems: four elements address the oceanographic science- and education-driven operations of the OOI integrated observatory, and two elements provide core infrastructure services for the distributed, message-based, service-oriented integration and communication infrastructure, as well as the virtualization of computational and storage resources. All OOI functional capabilities and resources represent themselves as services to the observatory network, with precisely defined service access protocols based on message exchange. This paper presents an overview of the OOI services and focuses on the strategy for service-oriented integration and the publish-subscribe model for communication.**

I. INTRODUCTION

The US National Science Foundation is initiating a transformation of ocean science with the *Ocean Observatories Initiative (OOI)* [1]. The OOI is designed to provide new, persistent, interactive capabilities for ocean science, and has a global physical observatory footprint. The *OOI Integrated Observatory* comprises Regional Scale Nodes (RSN) and Coastal/Global Scale Nodes (CGSN) providing cabled and buoy observatories with mobile instrument platforms, respectively.

The *OOI Cyberinfrastructure (CI)* [7] constitutes the integrating element of the OOI Integrated Observatory. It links and binds the physical observatory, computation, storage and network infrastructure into a coherent system-of-systems. The core capabilities and the principal objectives of the OOI Integrated Observatory are collecting real-time data, analyzing data, modeling the ocean on multiple scales and enabling adaptive and interactive experimentation within the ocean. A traditional data-centric CI, in which a central data management system ingests data and serves them to users on a query basis, is not sufficient to accomplish the range of tasks ocean scientists will engage in when the OOI is implemented. Instead, a highly distributed set of capabilities are required that facilitate:

- End-to-end data preservation and access,
- End-to-end, human-to-machine and machine-to-machine control of how data are collected and analyzed,
- Direct, closed loop interaction of models with the data acquisition process,
- Virtual collaborations created on demand to drive data-model coupling and share ocean observatory resources (e.g., instruments, networks, computing, storage and workflows),
- End-to-end preservation of the ocean observatory process and its outcomes, and
- Automation of the planning and prosecution of observational programs.

The OOI CI provides the software services and user interfaces to support these applications [13]; in addition it provides the underlying integration infrastructure [14] consisting of message-based communication, governance and security frameworks, similar to the role of the operating system on a computer. The CI also provides the mechanisms to execute distributed processes anywhere in the network and connect then into a coherent system of systems.

Section II describes the OOI Integrated Observatory Services and their Architecture from a high-level view. Section III describes the Common Operating Infrastructure (COI) subsystem as the central infrastructure component providing communication, governance and security to the

Integrated Observatory. In particular, we highlight the importance of the COI Messaging Service as the integration framework for the system-of-systems. Section IV provides a brief summary.

II. Integrated Observatory Services

Fig. 1 shows a schematic overview of the main properties of the Integrated Observatory functional design as provided by the Cyberinfrastructure component. The primary goal is to support the activities and applications of:

- Scientific Investigation, supporting researchers in the study of environmental processes though observations, simulation models and expressive analyses and visualizations, with results that directly feed back to improve future observations.
- Education and Participation, supporting education application developers, educators and the general public for accessing and understanding OOI resources in ways suitable for specific target audiences.
- Community Collaboration, enabling OOI users to share knowledge and resources, and to work together in project settings and ad hoc communities.

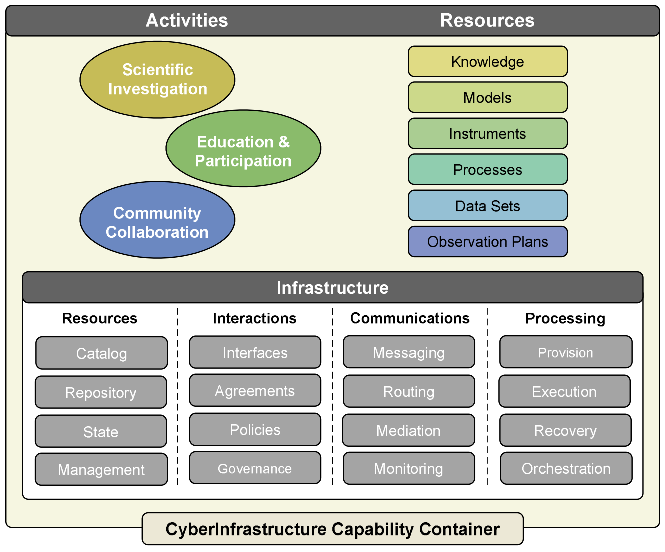


Fig. 1. OOI activities, resources and infrastructure

In support of these activities, a variety of Integrated Observatory resources of different type and purpose need to be administered, including:

- Observation Plans, providing activity sequences, service agreements and resource allocations for observational campaigns, and similar templates for event-response behaviors;
- Data Sets, representing observational and derived data and data products in the form of data archives and real-time continuous data streams;
- Processes, representing data collection and processing workflows that arrange multiple steps involving multiple actors and resources;
- Instruments and marine observatory infrastructure elements, such as telemetry systems, GPS and data loggers;
- Models, including numerical ocean forecast models and their configurations, as well as other analysis and event detection processes;
- Knowledge, representing all metadata, ancillary data, analysis results, association and correspondence links between resources, and knowledge captured in ontologies for semantic mediation purposes.

The support for these activities and resources rests on a collection of infrastructure services that provide resource management, interaction, communication and process execution. The *CI Capability Container* (see Fig. 1 and Fig. 2) is the extensible, deployable base unit of CI capabilities. It hosts all CI application services in support of activities and resource, infrastructure components and local interfaces; and it makes them available throughout the Integrated Observatory network forming a distributed system-of-systems.

The Integrated Observatory's functional capabilities are structured into six services networks (i.e., subsystems): four elements that address the ocean and Earth science- and education-driven operations of the OOI integrated observatory, and two elements that provide core infrastructure services for the distributed, message-based, service-oriented integration and communication infrastructure and the virtualization of computational and storage resources.

The *Sensing and Acquisition* services network provides capabilities to interface with and manage distributed seafloor instrument resources, as well as provide quality control services. The *Data Management* services network provides capabilities to distribute and archive data, including cataloging, versioning, metadata management, and attribution and association services. The *Analysis and Synthesis* services network provides a wide range of services to users, including control and archival of models, data analysis and visualization, event detection services and collaboration capabilities to enable the creation of virtual laboratories and classrooms. The *Planning and Prosecution* services network provides the ability to plan, simulate and execute observation missions using taskable instruments; it is the CI component that turns the OOI into an interactive observatory.

The remaining two services networks are the *Common Execution Infrastructure* (CEI) and the *Common Operating Infrastructure* (COI). The CEI provides an elastic computing framework to initiate, manage and store processes that may range from initial operations on data at a shore station to the execution of a complex numerical model on the national computing infrastructure and on compute clouds. The COI provides core services to manage distributed shared resources in a policy-based framework, including a distributed service infrastructure for the secure, scalable and fault tolerant operation and federation of the operational domains of authority comprising the OOI. It includes capabilities to manage identity and policy, manage any resource's life cycle, as well as catalog and repository services for observatory resources. It also manages interactions with resources on an end-to-end basis. An efficient messaging and service bus that incorporates security and governance, and provides guaranteed delivery, lies at its heart. Service-orientation and messaging realize loose coupling of components, resulting in the flexibility and scalability that are key in such a complex large-scale system-of-systems. All OOI functional capabilities and resources represent themselves as services to the observatory network, with precisely defined service access protocols based on message exchange.

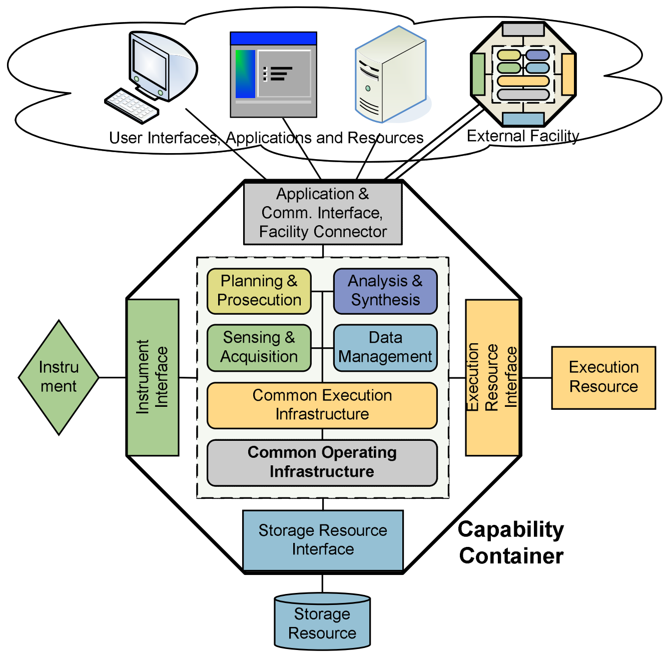

User Interfaces, Applications and Resources
External Facility
Application & Comm. Interface, Facility Connector
Planning & Prosecution
Analysis & Synthesis
Sensing & Acquisition
Data Management
Common Execution Infrastructure
Common Operating Infrastructure
Instru ment
Instrument Interface
Execution Resource Interface
Execution Resource
Storage Resource Interface
Capability Container
Storage Resource

Fig. 2. Capability Container with external interfaces

Fig. 2 depicts a capability container, indicated by the octagon shape, with interfaces to local resources and to the network environment. Local resources include physical resources such as instruments (sensors) and marine observatory infrastructure, storage resources such as disks and network drives, and computing resources such as grid nodes, cloud computing instances, and CPUs on mobile platforms such as AUVs (Autonomous Underwater Vehicles). Capability container can also be connected to user interfaces, external applications and to capability containers in different, independent facilities that have their own domains of authority and operation.

No matter where deployed, the capability container provides all of the infrastructure and application support required for an installation site within the OOI Integrated Observatory network. The capability container hosts the six services networks and their resource interfaces as depicted in the figure. The footprint of a capability container can vary depending on the resource constraints of its hosting environment. The selection of functional capabilities present in a specific capability container depends on the respective needs and resource availability at this specific location in the network. For instance, on an intermittently-connected instrument platform, instrument access, data acquisition and data buffering capabilities provided by the Sensing and Acquisition services are required, while the Analysis and Synthesis capabilities are not. In contrast, at the core installation sites, data processing, numerical model integration and event response behaviors need to be present.

III. Common Operating Infrastructure

The *Common Operating Infrastructure (COI)* [14] provides the integration fabric that enables subsystem services to be composed to manage complex interactions. The *Messaging Service* of COI provides dynamic routing and interception capabilities, a publish-subscribe [11] model for conversations, and reliable storage and delivery of messages to intended recipients across the network.

Rich Service Architecture

The COI architecture is based on the *Rich Services pattern* [4] a type of Service-Oriented Architecture (SOA) that provides decoupling between concerns and allows for hierarchical service composition. As depicted in Fig. 3, a Rich Service comprises several entities: (a) the *Service/Data Connector*, which serves as the sole mechanism for interaction between the Rich Service and its environment, (b) the *Messenger* and the *Router/Interceptor*, which together form the communication infrastructure, and (c) the constituent *Rich Services* connected to the Messenger and Router/Interceptor that encapsulate various application and infrastructure functionalities.

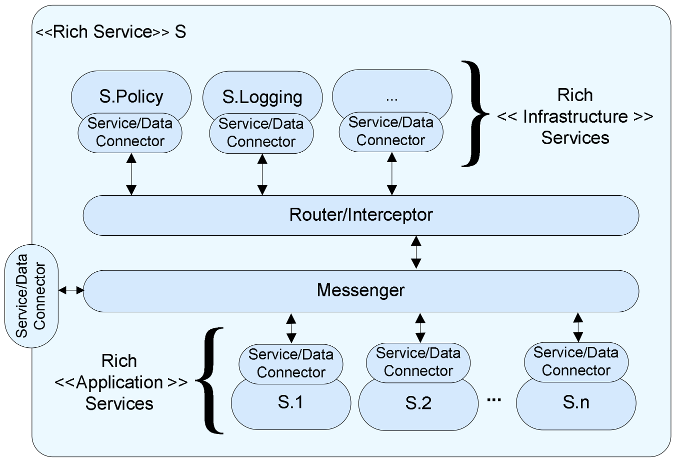


Fig. 3. Rich Services pattern

To address service integration, this architecture is organized around a message-based communication infrastructure. The

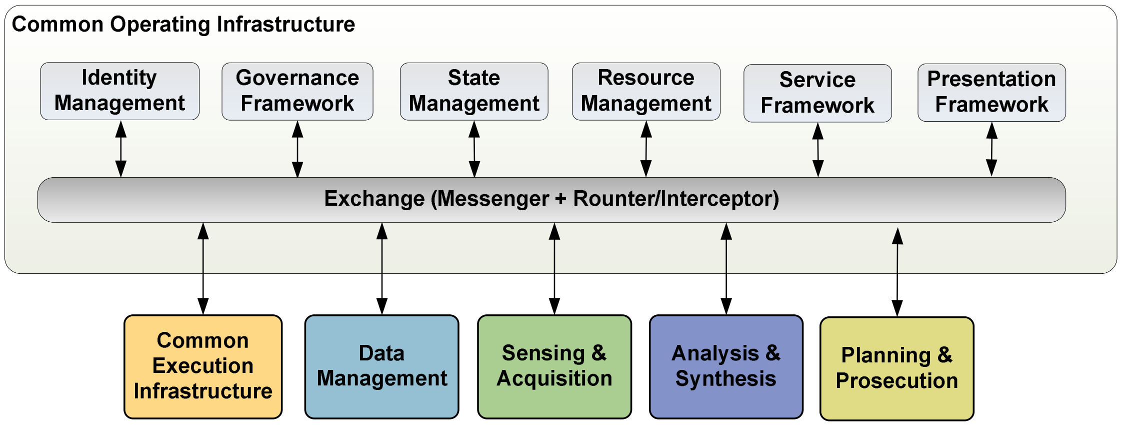


Fig. 4. Common Operating Infrastructure services

Messenger is responsible for message transmission between communication endpoints. By providing a means for asynchronous messaging, the Messenger supports the decoupling of Rich Services. The Router/Interceptor manages the interception of messages placed on the Messenger and their routing. This is useful for the injection of policies governing the integration of a set of services. The Service/Data Connector encapsulates and hides the internal structure of the connected Rich Service, and exports only the description and interfaces that the connected Rich Service needs to be visible externally. The communication infrastructure is only aware of the Service/Data Connector, and does not need to know any other information about the internal structure of the Rich Service.

Fig. 4 shows the Rich Services pattern applied to the COI architecture; the other five services networks are encapsulated as Rich Services connected to the COI messaging infrastructure (i.e., the *Exchange*). This shows the central and integrative role of the COI for the entire Integrated Observatory system-of-systems. The top of the figure depicts the infrastructure services that the COI provides to all subsystems. The COI ensures identity management, pervasive and consistent governance and policy enforcement, state management and resource management. It also enables subsystem services to be composed to handle complex interactions, and manages the overall service orchestration. The Router/Interceptor allows for flexible composition between the infrastructure and application services. In this way, there is a clear separation between the business logic and its external constraints. At all abstraction levels, infrastructure services plugged into the Exchange can modify the interaction patterns by re-routing, filtering, or modifying exchanged messages. This feature enables the validation and signing of messages, and the injection of policies governing the integration of a set of services.

The Rich Services integration strategy enables constituent subsystems to evolve independently from the composite system. Subsystem functionality is exposed to the OOI network as services with defined access interfaces, and the only way of interacting within the OOI network is through messages. Service-orientation and messaging realize loose coupling of components, resulting in flexibility and scalability. The complexity of such a large-scale system becomes manageable through separate concentration on each concern. Each subsystem focuses on the services that it enables and assumes that all of the infrastructure services are in place. For example, when designing the Sensing and Acquisition subsystem, the architecture team emphasizes concerns related to instrument control and data acquisition. Instruments can belong to individuals or the marine operators, while all of the deployment platforms are under the marine operator's authority domain. However, since governance is managed seamlessly by infrastructure services, and can be abstracted when designing the Sensing and Acquisition services, these issues are not of concern to the Sensing and Acquisition service developers.

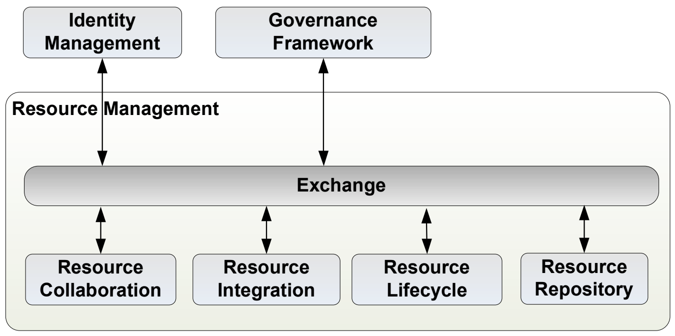


Fig. 5. COI Resource Management services

Each service of Fig. 4 is further decomposed according to the Rich Services pattern. For instance, Fig. 5 shows the internal decomposition for the *Resource Management* services. The *Resource Repository* service provides references to all resources known to the OOI CI. Through the *Resource Integration* service, resources can participate in interaction patterns implemented by OOI services (e.g., a storage resource may be used to record states of various services). The *Resource Collaboration* service provides the collaboration framework between different facilities and the sharing of resources within the OOI federation. The Resource Lifecycle service provides the means to track and manage resources throughout their entire lifecycle from development to decommissioning.

The Rich Services architecture provides resource location independence while user applications are shielded from the complexity of the system and the location of resources. The COI subsystem provides the Resource Management services that enable seamless use of resources across the entire Cyberinfrastructure. Via seamless integration of identity and governance services, the COI architecture supports the deployment, operation, and distributed management of thousands of independently-owned resources of various types (e.g., instruments, processes, numerical models and simulations) across a core infrastructure operated by independent stakeholders, where each stakeholder has different policies.

The COI Messaging Service (Exchange)

The *Exchange* (i.e., the COI Messaging Service or the Messenger and Router/Interceptor in the Rich Services architecture) is the central integrating element of the COI. It provides access to the communication mechanisms of *Exchange Spaces* and *Exchange Points* throughout the system-of-systems, abstracting from the physical communication infrastructure across multiple domains of authority. Client applications may publish messages on Exchange Points within Exchange Spaces. An Exchange Space represents a "community of interest" that collects and controls all of the Exchange Points in its scope and enforces policy of use for a registered set of users and applications. An Exchange Point is represented through a set of named exchanges on one or multiple AMQP [2] message brokers. Thereby, the Exchange provides a comprehensive, uniform view of a federation of message brokers: from the point of view of a publish/subscribe client (i.e., producers and consumers of messages), the fact that the messaging system is built as a federation of independent message brokers and not as a single broker is hidden.

The CI integration strategy determines how individual software components integrate into the system-of-systems through a message-broker integration infrastructure. The communication system of the OOI CI applies messaging as the central paradigm of inter-application information exchange, realizing the *Messaging Service*, the integrating element of all services. It is part of the *Common Operating Infrastructure (COI)*, the subsystem that provides the full set of integration frameworks and services (see [14]).

Message-oriented middleware (MOM) (see [6], [9]) is based on the concept of a message as the exclusive means of information exchange between the distributed components of a system. All information that is passed between two components or services is contained in messages exchanged asynchronously (i.e., non-blocking) over a communication infrastructure. The sender of a message does not wait for the message to be delivered or returned; it only waits for the MOM to acknowledge receipt of the message. Delivering messages to recipients utilizes the concept of queues. An application component in a message-oriented architecture only knows the incoming queues that it receives messages from as well as the outgoing queues it delivers messages to, plus the message formats that pertain to these queues. The MOM provides the capability for system integrators to connect these queues to known endpoints (i.e., addresses) in the network; consequently it manages routing, reliable storage and delivery of messages to intended recipients across the network. Standardization is on the way for the underlying message wire transport protocol: the Advanced Message Queuing Protocol (AMQP) [2] defines the interactions of a message broker with its clients, promising interoperability between message brokers of different provenance.

The left part of Fig. 6 depicts the fundamentals of the CI Messaging Service. Message brokers are the central infrastructure elements, represented as Exchange Points to all clients, responsible for the routing and delivery of messages.

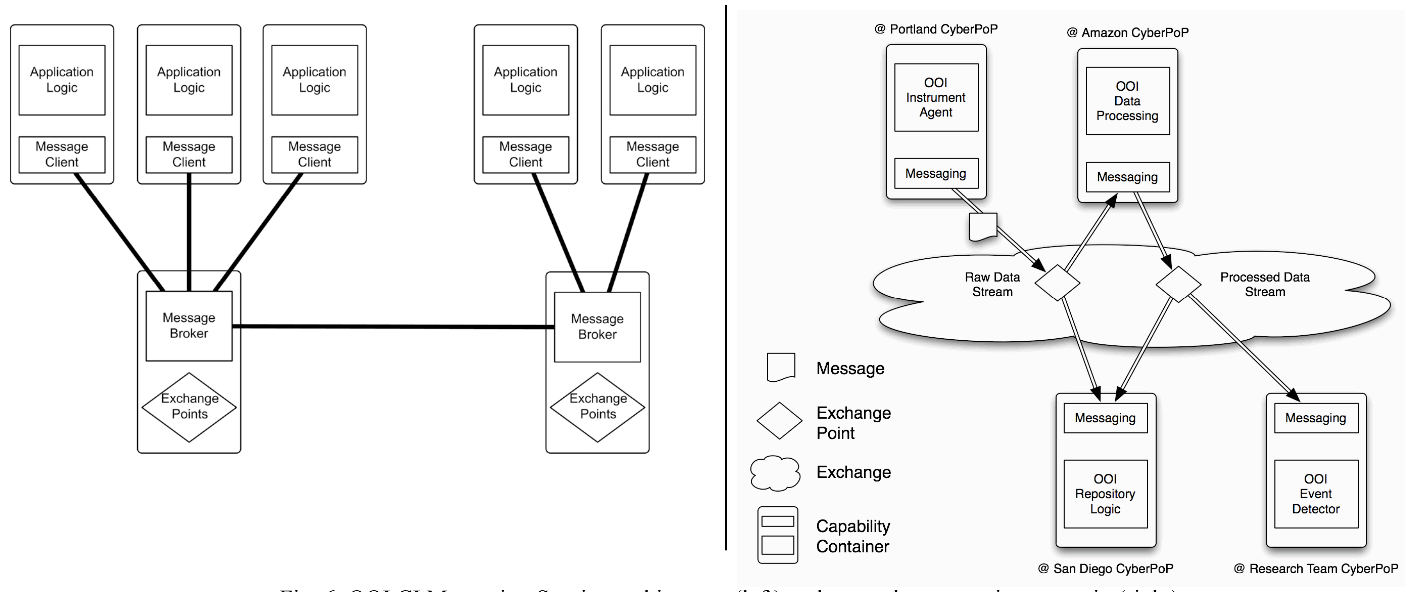


Fig. 6. OOI CI Messaging Service architecture (left) and exemplar messaging scenario (right)

Message Clients provide the interfaces to the application logic.

The right part of Fig. 6 provides an exemplar application scenario within the OOI CI. Capability containers host the application logic that interconnects using the Messaging Service. This is exemplified through an Instrument Agent publishing a raw data stream on an Exchange Point (a queue) via messaging. Any number of consumers may choose to subscribe to such an exchange point. In the example, the data processing application as well as the data repository will receive the published messages. A data stream is a continuous series of related self-contained messages on a given exchange point. There is a second exchange point for another data product containing processed data that is consumed by an event detector process. The physical deployment of all applications is irrelevant. The Exchange realizes all connectivity.

Fig. 7 depicts an exemplar scenario of how service clients can adapt to the Messaging Service; we have implemented this in current prototypes [15]. Services are identified by name within the Exchange network throughout the entire system-of-systems. Services are part of distributed applications; the distributed service interaction protocol at every (service) endpoint is implemented by a specialized protocol adapter. Such protocol adapters are instantiated for each conversation instance (see below for further details) through protocol factories; the protocol adapters provide the binding element to the actual service application and its functionality. A typical mechanism of implementing protocol adapters is using Finite State Machines (FSM). FSMs represent each distinguishable protocol condition as a separate state, with defined transitions between states when messages are sent or received, leading to very predictable and robust distributed implementations. We have prototyped several interaction styles between service applications, including direct point-to-point interaction, topic based publish/subscribe fan-out queues and worker queues that facilitate reliable load-balanced applications. The Messaging Service hides the fact that service applications are connected to different message brokers that are operated in different domains of authority.

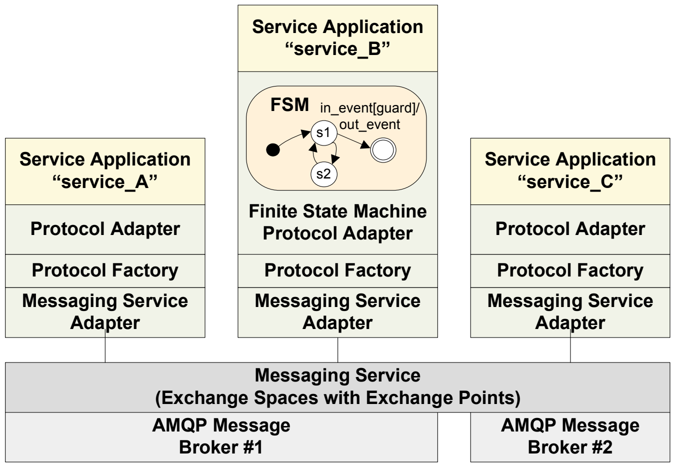

Service Application "service_B"
FSM
in_event[guard]/ out_event
s1
s2
Finite State Machine Protocol Adapter
Protocol Factory
Messaging Service Adapter
Service Application "service_A"
Protocol Adapter
Protocol Factory
Messaging Service Adapter
Service Application "service_C"
Protocol Adapter
Protocol Factory
Messaging Service Adapter
Messaging Service (Exchange Spaces with Exchange Points)
AMQP Message Broker #1
AMQP Message Broker #2

Fig. 7. Messaging Service and service client adapters prototype

Communities and Agents

Our approach in distributed computing is based on the premise that independent entities interact in order to pursue shared goals. Entities can represent users, processes, resources and communities.

Entities in the system are represented by their *agents*. Each entity (or their agent on their behalf) can form any number of relationships with other entities. Relationships are based on mutual (bilateral) agreements between two entities, the results of a successful negotiation. Each entity tracks the consequences (i.e., commitments [9], [16]) of such agreements (i.e., contracts) with other entities. Each observable atomic action of an entity, such as sending a message, that causes a side effect leads to a change and reevaluation of the aggregate set of commitments of the entity towards other entities.

Entities communicate and collaborate within communities. A community is a specific type of entity in itself. Communities serve multiple purposes in our architecture, including providing a backstop for contracts, providing a locus for naming, and providing a venue to share resources in some uses including infrastructure. A community is represented by a specification that defines the rules for joining the community. Joining a community requires accepting the rules of the community, and the community will provide the registrant entity with a local name and address.

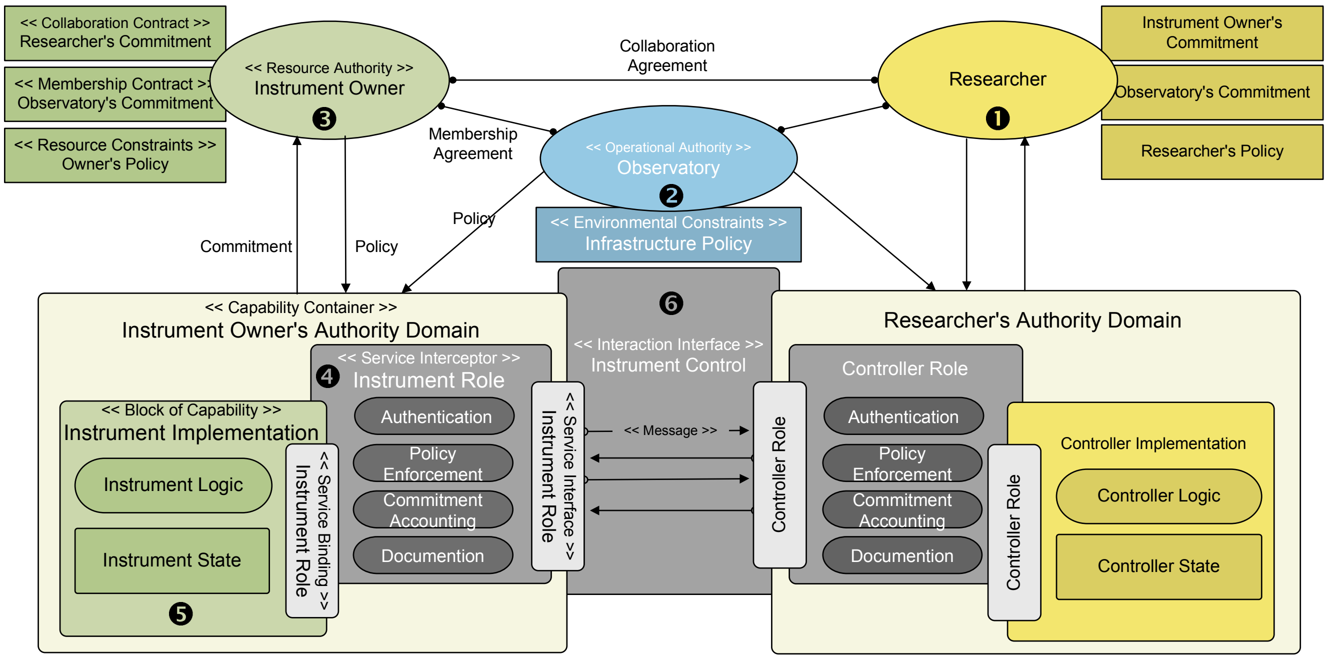
Entities may request to enroll (i.e., participate) in communities or can be invited by other member entities into the community. Enrollment is a symmetric process of negotiation. Entities negotiate the conditions under which they participate in the community and vice versa. If agreement is reached, the resulting contract builds the basis for relations with other community members.

Communities can form relationships with other communities, enabling the members of one community to interact with the members of another community, instituting the specifications of both communities. By contract, the community members are bound to the community specification with its rules, so there is no need for explicit compliance checking (i.e., policy enforcement) and members can interact directly. There might be an imposed requirement for members to leave behind audit trails for later evaluation, same as a tax rule not being directly enforced with every transaction, but which may be audited for compliance to the "state" community tax rules later for each member taxpayer.

We call the set of rules that communities (or other entities) impose *policy*. Policy to access a resource entity for instance might be an aggregate of many rules, such as the resource owner's rules, the community's rules, and any underlying obligations as consequence of membership.

Conversation Management

Communication between two entities occurs as part of a *conversation*. A conversation presumes a contract is in place between the two entities intending to converse. This contract



<< Collaboration Contract >> Researcher's Commitment
<< Membership Contract >> Observatory's Commitment
<< Resource Constraints >> Owner's Policy
<< Resource Authority >> Instrument Owner
❸
Collaboration Agreement
Membership Agreement
Researcher
❶
Instrument Owner's Commitment
Observatory's Commitment
Researcher's Policy
<< Operational Authority >> Observatory
❷
<< Environmental Constraints >> Infrastructure Policy
Commitment
Policy
Policy
<< Capability Container >> Instrument Owner's Authority Domain
❻
<< Interaction Interface >> Instrument Control
Researcher's Authority Domain
<< Service Interceptor >> Instrument Role
❹
Authentication
Policy Enforcement
Commitment Accounting
Documention
<< Block of Capability >> Instrument Implementation
Instrument Logic
Instrument State
❺
<< Service Binding >> Instrument Role
<< Service Interface >> Instrument Role
<< Message >>
Controller Role
Controller Role
Authentication
Policy Enforcement
Commitment Accounting
Documention
Controller Role
Controller Implementation
Controller Logic
Controller State

Fig. 8. Collaboration and policy framework example

must include the common knowledge of an interaction pattern that provides a template for the conversation, with the conversation being an instantiation of the pattern. The actual interaction as part of the conversation must comply with the template of the interaction pattern. Each interaction (sending and receipt of a message) potentially causes change in the set of commitments related to the conversation and, thus, indirectly to the commitments between the two entities. Interaction patterns are thereby distributed Assumption/Commitment specifications, in particular also for policy. Each entity can independently monitor the fulfillment of the interaction pattern and contract for the other entity and for itself (and initiate protective or compensating action otherwise). Each party would thus update its commitment store based on each message it sends or receives. Each entity can engage in as many different conversations with different (or the same) entities concurrently as it likes. At any given instant, the effective set of commitments from the point of view of the entity is defined; each interaction can be traced back to a conversation.

We specify interaction patterns using Message Sequence Charts (MSCs, see [7], [9], [12]). We also define a language for commitments that are made and released for each interaction in an interaction pattern. We provide a logical framework to reason over the aggregate set of commitments over time and deduce any implications. Currently, we use a rules engine to implement such a mechanism.

The COI provides collaboration, agreement support, and policy enforcement capabilities. Fig. 8 illustrates this pattern for the base case of a single service provider (instrument owner) and consumer (researcher). The pattern generalizes to arbitrary numbers of participants in a service orchestration. Conceptually, the example captures the establishment of a service agreement between two parties; for example, this could unfold between a regional cabled observatory (service provider) and a buoy-based global observatory (service consumer). Each one of the parties has established contractual commitments with their respective user communities, including membership agreements. Upon the establishment of mutual commitments, a contract between the two parties is in place. Further, each party operates under its own set of policies. The negotiation and contracting process, as well as the actual service usage, leads to an interaction pattern between the two parties that is constrained by the contractual commitments and policy declarations of both parties.

Because each Capability Container is equipped with plug-ins for orchestration, governance, policy enforcement, and monitoring/audit, the deployment mapping for the collaboration and policy framework is straightforward: the corresponding interaction interface is stored and accessed CI-wide. Each party's Capability Container orchestration component executes the projection of the interaction pattern on their respective roles to participate in the overall collaboration. The governance and policy constraints are extracted from the interaction interface and provided to the corresponding Capability Container plug-ins for monitoring and enforcement.

The COI, through the use of the CI capability container, factors out the common aspects of communication, state management, execution, governance, and service presentation to provide a highly scalable, secure and extensible model for managing user-defined collections of information and taskable resources. This ability to integrate resources of different types implemented by different technologies is the central value

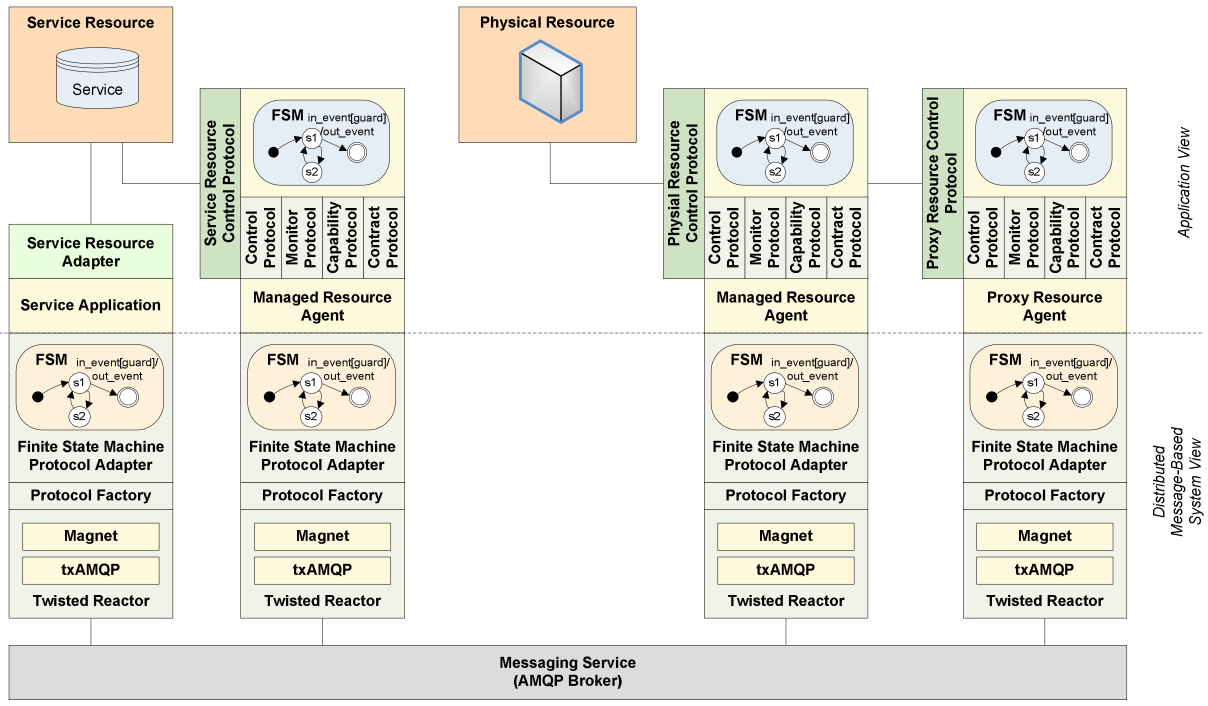


Fig. 9. Resource, resource agents and resource proxies connected to the Messaging Service

proposition of the architecture. It provides the basis for an integrated observatory network that will remain viable and pertinent over multiple decades.

Protocols are defined through interaction patterns. The interaction pattern (or projection thereof) represents the interaction interfaces of entities (i.e., components). The projection of a protocol on one party can be represented as a Finite State Machine (FSM). We use FSMs as protocol machines that bind the communication endpoint on an asynchronous reliable message-based system to the application logic. Fig. 7 shows the use of FSMs as protocol adapters for service applications involved in a conversation as defined by an interaction pattern.

Fig. 9 shows an exemplar scenario for the application of agents for the management of physical resources such as sensors, and of services in a distributed environment. Agents interact via the Messaging Service. Services themselves use the Messaging Service for inter-service conversations as explained above. In this case, the services' agents provide the management and control for the service, such as starting/stopping the service and granting access. Finite State Machines as protocol adapters ensure that the agents and service protocols are always in a consistent distributed state, ensuring robustness of the entire system. Service protocol adapters provide access to the service; Managed Resource Agent protocol adapters provide access to the respective resource agents. Resource agents provide monitoring and control of resources, advertise and grant access to resource capabilities and manage the contractual relations and commitments of the resource to its environment on behalf of the resource. All these agent interactions occur in form of conversations based on defined interaction patterns. Proxy Resource Agents provide similar capabilities and interaction patterns but act as proxies or supervisors of Managed Resource Agents. Thereby, policy can be applied at various levels within the system through a chain of responsibility.

Distributed IPC Facility

We are currently investigating a special case of community called the Distributed Inter-Process Communication Facility (DIF) [5]. Entities, representing processes that require inter-process communication (IPC), enroll in this community and are assigned a name valid throughout the community as well as an address that the community uses internally to direct communication. The resources of the community are local endpoints of the DIF, which provide resource allocation (open/close a connection to another named endpoint) and read/write capabilities.

This DIF facility is intended to be the underlying distributed system primitive within the OOI system-of-systems. As is apparent, in conceptual terms, DIFs relate naturally to the notion of communities that we motivated in the foregoing. Other communities will be defined applying similar patterns for other purposes than communication, such as scalable, elastic computing environments, with entities including the requestors of a service and the responding nodes.

The power of the DIF model is that it can be stacked in order to increase scope. One DIF can leverage a lower level DIF for communication purposes and present a DIF facility of larger scope to its member entities. Thereby, the design of how to architect the communities becomes the driving element in the architecture of a distributed system. Any topology and architecture is possible here, exceeding pure layered architectures.

IV. Summary

The Ocean Observatories Initiative faces the enormous challenge of building a cohesive distributed system-of-systems that incorporates a large number of autonomous and heterogeneous systems, deals with instruments and computational resources of a wide range of capabilities, serves the needs of diverse stakeholders, and accommodates change over the timescale of decades. A carefully thought out architecture is key to addressing this challenge. We find that simplicity wins and a few core principles help us organize the OOI properly. These principles include (1) emphasizing loose coupling through message-based interactions; (2) recognizing the autonomy of the participants by modeling them as agents rather than as traditional objects or pure services; (3) identifying repeating structures (as evinced in our choice of Rich Services, Capability Containers, DIFs, and communities); and capturing and making explicit business-level interactions through first-class status for policy and governance.

Acknowledgment

The OOI Cyberinfrastructure program is funded through the JOI Subaward, JSA 7-11, which is in turn funded by the NSF contract OCE-0418967 with the Consortium for Ocean Leadership, Inc.

References

[1] Ocean Observatories Initiative (OOI). Program website, http://www.oceanleadership.org/ocean_observing/ooi

[2] Advanced Message Queuing Protocol (AMQP). AMQP Working Group Website http://www.amqp.org/

[3] Amazon.com, Amazon Web Services for the Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[4] M. Arrott, B. Demchak, V. Ermagan, C. Farcas, E. Farcas, I. H. Krüger, and M. Menarini. Rich Services: The Integration Piece of the SOA Puzzle. In Proc. of the IEEE International Conference on Web Services (ICWS), Salt Lake City, Utah, USA. IEEE, Jul. 2007, pp. 176-183.

[5] J. Day. Patterns in Network Architecture: A Return to Fundamentals. Prentice Hall, 2008.

[6] G. Banavar, T. Chandra, R. Strom and D. Sturman. A case for message oriented middleware. Proc. of the 13th International Symposium on Distributed Computing, pp. 1–18, 1999.

[7] M. Broy, I. H. Krüger, and M. Meisinger. A Formal Model of Services. ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 16, no. 1, p. 5, Feb. 2007

[8] A. Chave, M. Arrott, C. Farcas, E. Farcas, I. Krueger, M. Meisinger, J. Orcutt, F. Vernon, C. Peach, O. Schofield, and J. Kleinert. Cyberinfrastructure for the US Ocean Observatories Initiative: Enabling Interactive Observation in the Ocean. In Proc. IEEE OCEANS'09 Bremen, Germany. IEEE Ocean Engineering Society, May 2009.

[9] A.K. Chopra and M.P. Singh. An Architecture for Multiagent Systems: An Approach Based on Commitments. Proceedings of the AAMAS Workshop on Programming Multiagent Systems (ProMAS). May 2009

[10] B. Demchak, V. Ermagan, E. Farcas, T.-J. Huang, I. Krüger, and M. Menarini, "A Rich Services Approach to CoCoME," The Common Component Modeling Example: Comparing Software Component Models, A. Rausch, R. Reussner, R. Mirandola, and F. Plasil (Eds.), Lecture Notes in Computer Science, no. 5153, ch. 5, pp. 85-115, Berlin/Heidelberg: Springer-Verlag, Aug. 2008

[11] P.T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Tech. Rep. DSC ID:2000104, EPFL, January 2001.

[12] I. H. Krueger, M. Meisinger, and M. Menarini. Interaction-based Runtime Verification for Systems of Systems Integration. Journal of Logic and Computation, Nov. 2008

[13] OOI CI Integrated Observatory Applications Architecture Document, OOI controlled document 2130-00001, version 1-00, 10/28/2008, available at http://www.oceanobservatories.org/spaces/display/FDR/CI+Technical+File+Repository

[14] OOI CI Integrated Observatory Infrastructure Architecture Document, OOI controlled document 2130-00002, version 1-00, 10/24/2008, available at http://www.oceanobservatories.org/spaces/display/FDR/CI+Technical+File+Repository

[15] OOI CI Messaging Service Prototype. http://www.oceanobservatories.org/spaces/display/CIDev/Messaging+Service

[16] M.P. Singh. Semantical Considerations on Dialectical and Practical Commitments. Proceedings of the 23rd Conference on Artificial Intelligence (AAAI). July 2008, pp. 176-181.